README.md - Grip

FEUP_IART

Projeto 1 - Folding Blocks

Autores:

- Bernardo Moreira - up201604014
- Francisco Pereira - up201605306
- Filipe Nogueira - up201604129

PROJETO 1: FOLDING BLOCKS

Especificação do trabalho

- Neste projeto é pretendido implementar um algoritmo de inteligência artificial capaz de vencer uma série de niveis do jogo **Folding Blocks**. Este jogo inicialmente composto por um tabuleiro com uma ou mais peças, tem como objetivo, ocupar todos os espaços dos tabuleiros utilizando essas peças. Esta utilização é feita através de uma seleção da peça a jogar, seguida de uma transformação simétrica, isto é, cada movimento vai **duplicar** o tamanho da peça na direção escolhida.

Formulação do Problema

- Representação do estado : O tabuleiro do jogo vai ser representado por uma matriz com tamanho variável (varia conforme o nivel). Esta matriz é um `int[][]` cujo valor inicial para todas as posições é 0 representando assim os espaços vazios. Seguidamente, e consoante o nivel, são predefinidas peças para se jogar, por exemplo, no nivel 1 só é colocada uma peça na primeira posição, portanto `board[0][0] = 1`. Caso existam mais peças o id desta vai incrementando.

- Teste objetivo : O jogo acaba quando o tabuleiro não tiver espaços vazios. Ou seja, quando em todas as posições do tabuleiro x e y, `board[x][y]` seja diferente de 0.

- Operadores :

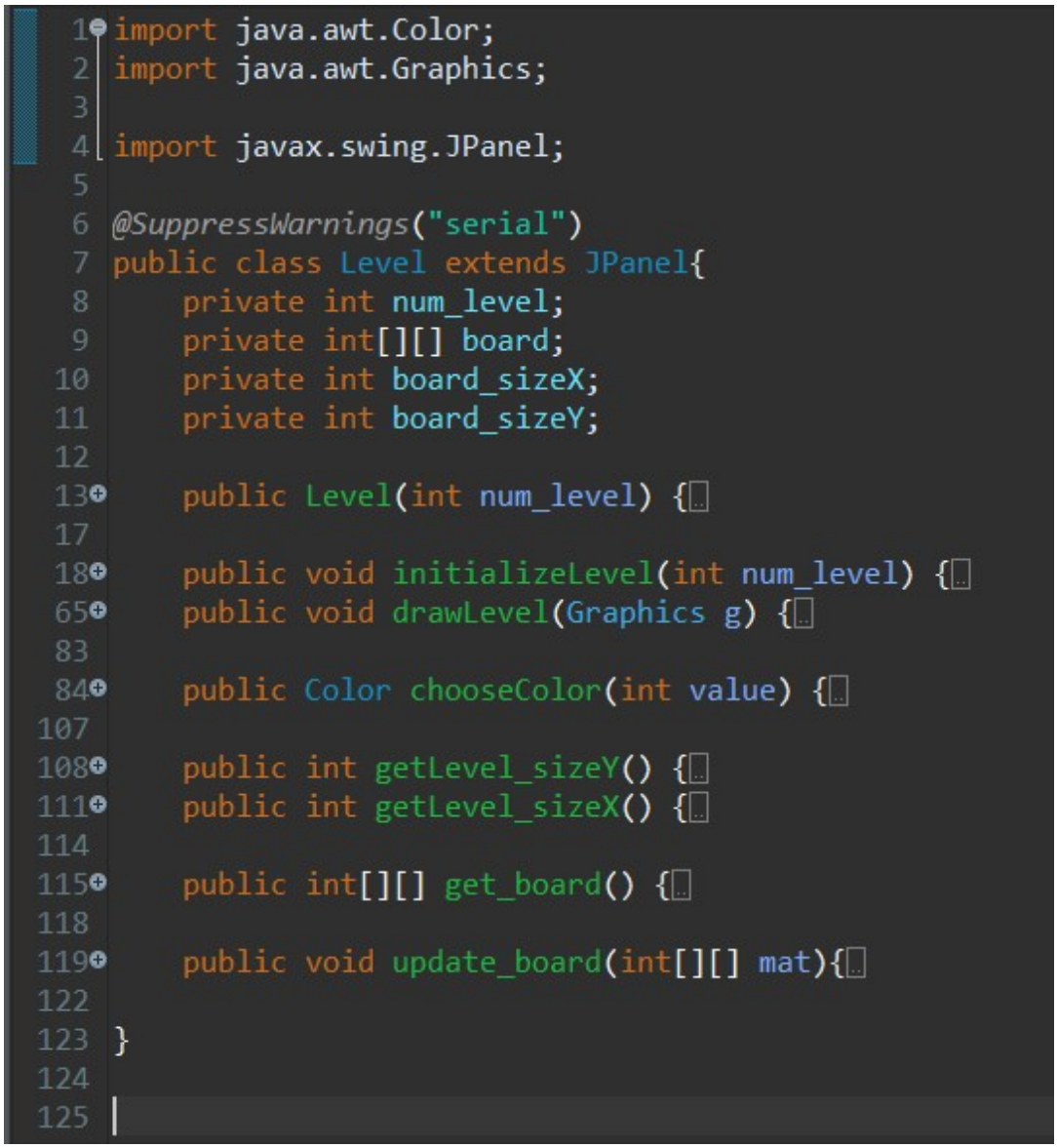
Nomes	Pré-condições	Efeitos	Custos
Dobrar para a Esquerda	Yb >= 0; Tab[xi][yb] = 0;	Dist = (yi - y_eixo) + 1; Yb = y_eixo - Dist; Tab[Xi][Yb] = ID;	1
Dobrar para a Direita	Yb < M; Tab[xi][yb] = 0;	Dist = (y_eixo - yi) + 1; Yb = y_eixo + Dist; Tab[Xi][Yb] = ID;	1
Dobrar para Cima	Xb >= 0; Tab[xb][yi] = 0;	Dist = (xi - x_eixo) + 1; Xb = x_eixo - Dist; Tab[Xb][Yi] = ID;	1
Dobrar para Baixo	Xb < N; Tab[xb][yi] = 0;	Dist = (x_eixo - xi) + 1; Xb = x_eixo + Dist; Tab[Xb][Yi] = ID;	1

    - ID - referente à peça a ser jogada;
    - Tab[ ][ ] - valor na posição de cada tabuleiro (pode ter os seguintes valores: 0 - se estiver livre / ID - numero referente à peça a ser jogada);
    - Xi / Yi - Coordenada referente à linha/coluna (respectivamente) do bloco atual;
    - Xb / Yb - Coordenada referente à linha/coluna (respectivamente) do bloco novo;
    - x_eixo / y_eixo - Coordenada referente à linha/coluna (respectivamente) do eixo de simetria;
    - Dist - distância ao eixo de simetria;
    - N x M - dimensões do tabuleiro de jogo (linhas/colunas);
- A heuristica será a distância até à solução. Por outras palavras, será a diferença entre o Tamanho do tabuleiro e o número de quadrados preenchidos

Implementação do jogo

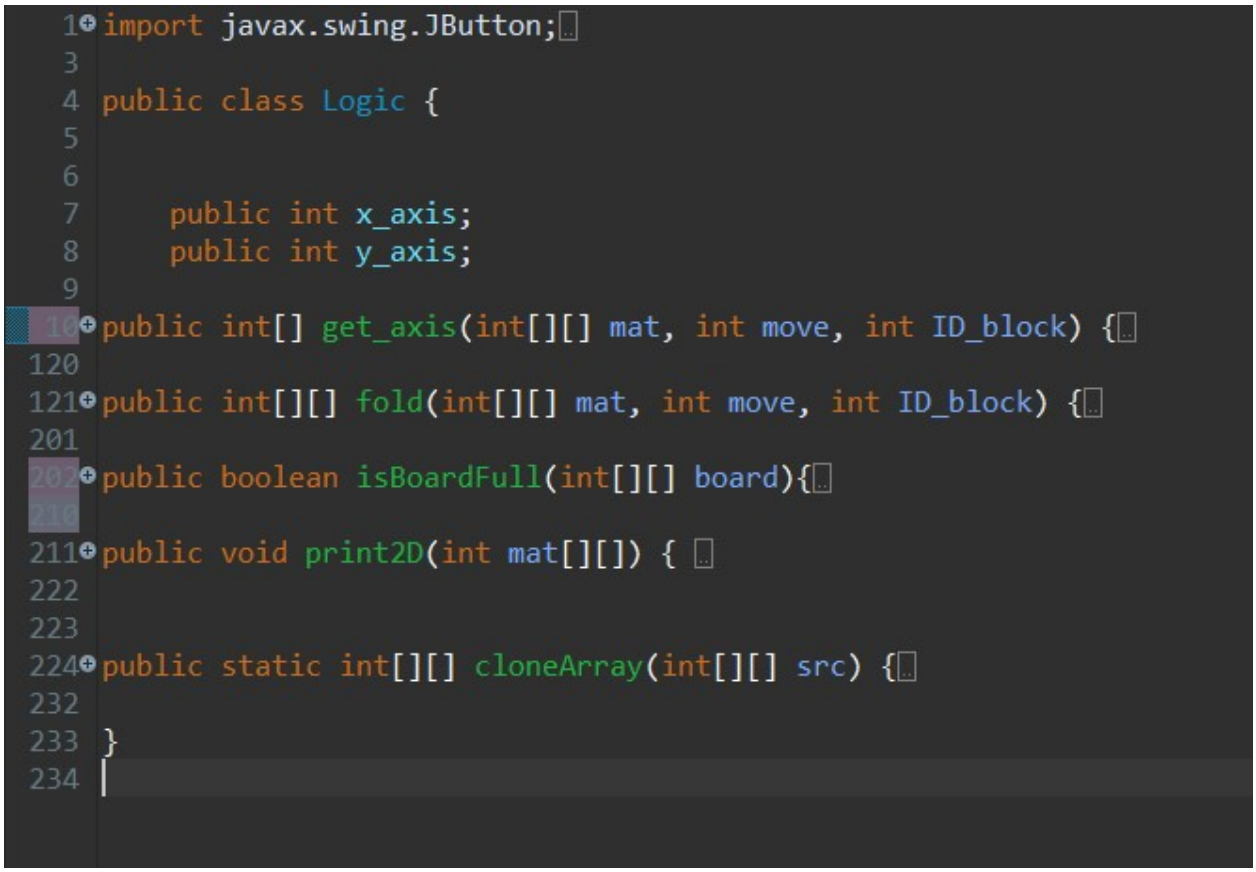
- Linguagem escolhida para desenvolvimento do código : **Java**
- Trabalho realizado em **Eclipse** & **VSCode**
- Código encontra-se dentro de uma pasta **src** contendo os seguintes ficheiros :
    - Main.java
    - Game.java
    - Level.java
    - Logic.java
- A classe Level trata de definir, e desenhar os niveis. Trata da atribuição de cores consoante os valores

da matriz e trata de atualizar a mesma.



```
import java.awt.Color;
import java.awt.Graphics;

import javax.swing.JPanel;

@SuppressWarnings("serial")
public class Level extends JPanel{
    private int num_level;
    private int[][] board;
    private int board_sizeX;
    private int board_sizeY;

    public Level(int num_level) {

    public void initializeLevel(int num_level) {
    public void drawLevel(Graphics g) {

    public Color chooseColor(int value) {

    public int getLevel_sizeY() {
    public int getLevel_sizeX() {

    public int[][] get_board() {

    public void update_board(int[][] mat){

}
```

- A classe Logic verifica a logica de jogo. Trata de fazer as jogadas, bem como as verificações de jogada e de fim de jogo.



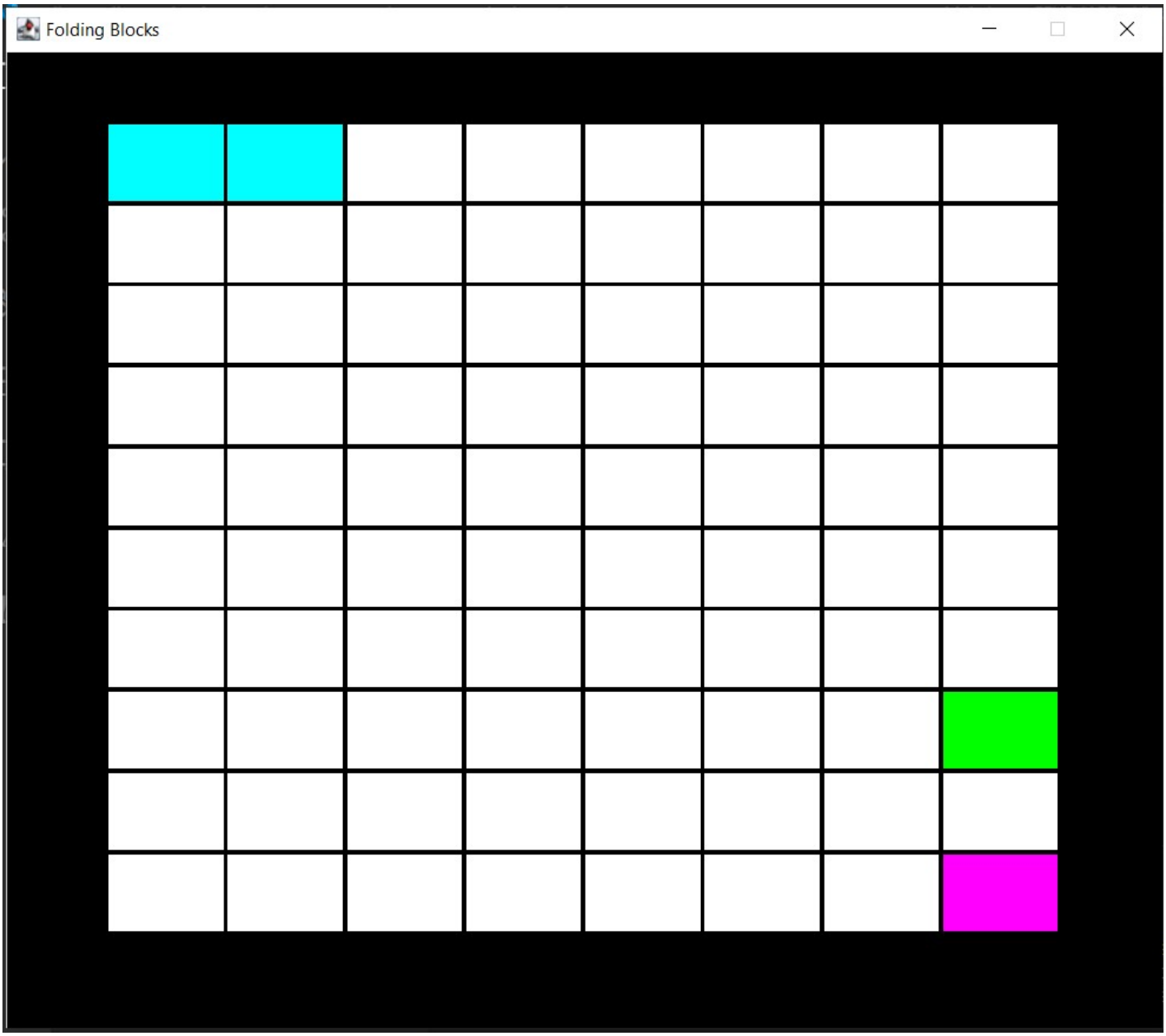
```
1  import javax.swing.JButton;
3
4  public class Logic {
5
6
7      public int x_axis;
8      public int y_axis;
9
10 public int[] get_axis(int[][] mat, int move, int ID_block) {
120
121 public int[][] fold(int[][] mat, int move, int ID_block) {
201
202 public boolean isBoardFull(int[][] board){
216
211 public void print2D(int mat[][]) {
222
223
224 public static int[][] cloneArray(int[][] src) {
232
233 }
234
```

- Finalmente, tanto a classe Main como Game servem para funcionalidades da interface. Permite a criação de uma janela para o jogo, bem como permite à classe Level que desenhe nesta mesma interface. Inicialmente implementamos também deteção de pressão de teclas, para testarmos as funcionalidades do jogo antes de implementar os algoritmos.

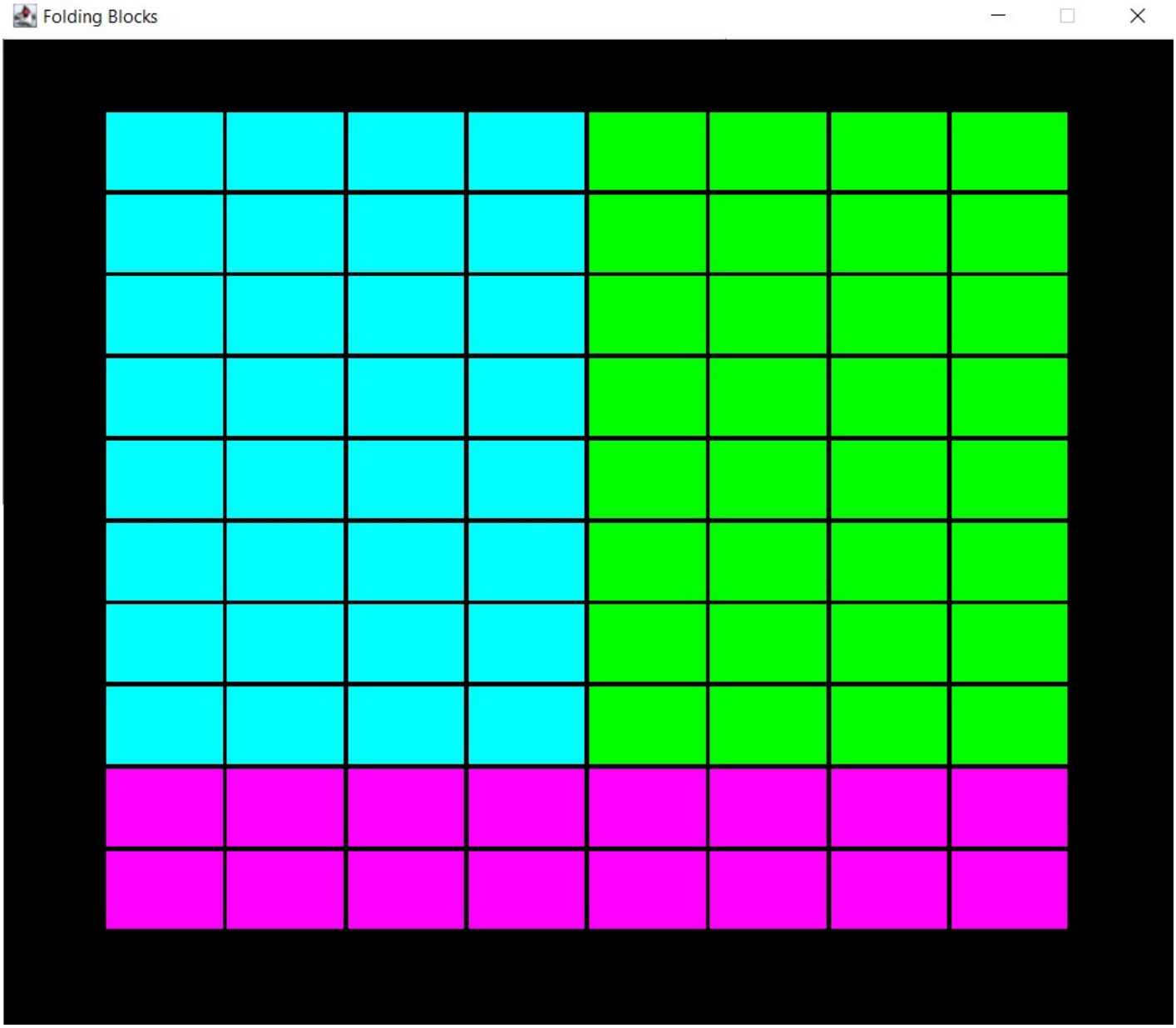
```java
import java.awt.Color;

@SuppressWarnings({ "serial", "unused" })

public class Game extends JPanel implements KeyListener, ActionListener {

    private Timer timer;
    private int num_level;
    private Level l;
    private int[][] mat;

    private int move;
    private int ID_block;
    public boolean right;
    public boolean left;
    public boolean up;
    public boolean down;
    public Logic functional;

    public Game() {

    public void paint(Graphics g) {

    public void actionPerformed(ActionEvent arg0) {

    public void keyPressed(KeyEvent e) {

    public void keyReleased(KeyEvent e) {

    public void keyTyped(KeyEvent arg0) {

}
```

```java
import javax.swing.JFrame;

public class Main {

    public static void main(String[] args) {

        JFrame obj = new JFrame();
        obj.setBounds(10, 10, 800, 700);
        Game game = new Game();
        obj.setTitle("Folding Blocks");
        obj.setResizable(false);
        obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        obj.add(game);
        obj.setVisible(true);
    }
}
```

Representação do estado inicial do nivel 6



Representação do estado final do nivel 6


Folding Blocks